Exercises: Abstraction

This document defines the exercises for "Java Advanced" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

1. Fill the Matrix

Filling a matrix in the regular way (**top to bottom** and **left to right**) is boring. Write two **methods** that **fill** a **matrix** of size **N x** N in **two** different **patterns.** Both patterns are described below:

Pattern A	Pattern B
1 5 9 13 2 6 10 14 3 7 11 15 4 8 12 16	1 8 9 16 2 7 10 15 3 6 11 14 4 5 12 13

Examples

Input	Output
3, A	1 4 7 2 5 8 3 6 9
3, B	1 6 7 2 5 8 3 4 9

Hints

- Make a different method for each pattern
- Make a method for printing the matrix

2. Matrix of Palindromes

Write a program to generate the following **matrix of palindromes** of **3** letters with **r** rows and **c** columns like the one in the examples below.

- **Rows** define the first and the last letter: row 0 → 'a', row 1 → 'b', row 2 → 'c', …
- **Columns + rows** define the middle letter:
  - column 0, row 0 → 'a', column 1, row 0 → 'b', column 2, row 0 → 'c', …
  - column 0, row 1 → 'b', column 1, row 1 → 'c', column 2, row 1 → 'd', …

Input

- The numbers **r** and **c** stay at the first line at the input.
- **r** and **c** are integers in the range [1…26].
- **r + c** ≤ 27

Examples

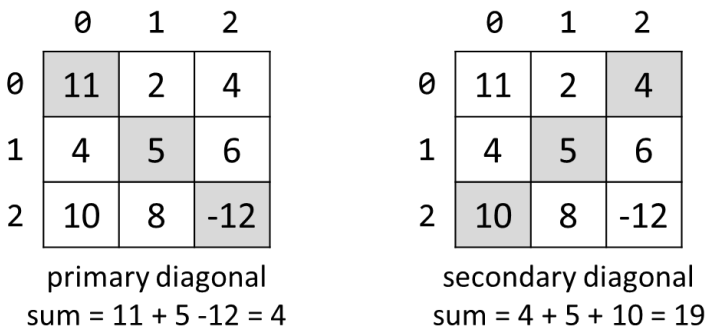
Input	Output
4 6	aaa aba aca ada aea afa bbb bcb bdb beb bfb bgb ccc cdc cec cfc cgc chc ddd ded dfd dgd dhd did
3 2	aaa aba bbb bcb ccc cdc

Hints

- Use two nested loops to generate the matrix.
- Print the matrix row by row in a loop.
- Don't forget to pack everything in methods.

3. Diagonal Difference

Write a program that finds the **difference between the sums of the square matrix diagonals** (absolute value).



Input

- The **first line** holds a number **n** – the size of matrix.
- The next **n lines** hold the **values for every row** – **n** numbers separated by a space.

Examples

Input	Output	Comments
3 11 2 4 4 5 6 10 8 -12	15	**Primary diagonal:** sum = 11 + 5 + (-12) = 4 **Secondary diagonal:** sum = 4 + 5 + 10 = 19 **Difference:** |4 - 19| = 15

Hints

- Use a **single** loop **`i`** = **`[1 … n]`** to sum the diagonals.
- The **primary diagonal** holds all cells {**`row`**, **`col`**} where **`row`** == **`col`** == **`i`**.
- The **secondary diagonal** holds all cells {**`row`**, **`col`**} where **`row`** == **`i`** and **`col`** == **`n-1-i`**.

4. 2x2 Squares in Matrix

Find the count of **2 x 2 squares of equal chars** in a matrix.

Input

- The matrix size is given at the first row (**rows** and **columns**).
- Matrix characters come at the next **rows** lines (space separated).

Examples

Input	Output	Comments
3 4 A B B D E B B B I J B B	2	Two 2 x 2 squares of equal cells: A **B B** D E **B B** B I J B B A B B D E B **B B** I J **B B**
2 2 a b c d	0	No 2 x 2 squares of equal cells exist.

Hints

- Check all possible 2 x 2 squares for equal elements.
- Make sure your program wouldn't throw an **IndexOutOfBoundsException()**

5. Maximal Sum

Write a program that reads a rectangular integer matrix of size **N x M** and finds in it the square **3 x 3** that **has maximal sum of its elements**.

Input

- On the first line, you will receive the rows **N** and columns **M**.
- On the next **N lines** you will receive **each row with its elements**.

Print the **elements** of the 3 x 3 square as a matrix, along with their **sum**. See the format of the output below:

Examples

Input	Output	Comments
4 5 1 5 5 2 4 2 1 4 14 3 3 7 11 2 8 4 8 12 16 4	Sum = 75 1 4 14 7 11 2 8 12 16	1 5 5 2 4 2 **1 4 14** 3 3 **7 11 2** 8 4 **8 12 16** 4

6. Sequence in Matrix

You are given a matrix of strings of size **N x M**. Sequences in the matrix are defined as sets of several neighbour elements, located on the same **diagonal, line or column** . Write a program that finds the longest sequence of equal strings in the matrix.

If there are **two sequences with the same length**, **print the one that you found last (Check the diagonals first, then the lines and last the columns)**

Input

- On the first line, you will receive the rows **N** and columns **M**.
- On the next **N lines** you will receive **each row with its elements**.

Examples

Input	Output	Comments
3 4 ha fifi ho hi fo ha hi xx xxx ho ha xx	ha, ha, ha	**ha** fifi ho hi fo **ha** hi xx xxx ho **ha** xx
3 3 s qq s pp pp s pp qq s	s, s, s	s qq **s** pp pp **s** pp qq **s**

7. Collect the Coins

Working with multidimensional arrays can be (and should be) fun. Let's make a game out of it.

You receive the layout of a **board** from the console. Assume it will always have **4 rows** which you'll get as strings, each on a separate line. Each character in the strings will represent a **cell** on the board. Note that the strings may be of different lengths.

You are the player and start at the **top-left** corner (that would be position **[0, 0]** on the board). On the fifth line of input you'll receive a string with movement commands which tell you where to go next, it will contain only these four characters – '**>**' (move **right**), '**<**' (move **left**), '**^**' (move **up**) and '**v**' (move **down**).

You need to keep track of two types of events – collecting **coins** (represented by the symbol '**$**', of course) and hitting the **walls** of the board (when the player tries to move off the board to **invalid** coordinates). When all moves are over, **print the amount of money** collected and the **number of walls hit**.

Examples

Input	Output	Comments
Sj0u$hbc $87yihc87	Coins = 2	Starting from (0, 0), move down (coin), twice right, up, up again (wall), three times right

Ewg3444 $4$$ V>>^^>>>VVV<<	Walls = 2	(coin on second move), twice down, down again (wall), twice to the left – game over (no more moves). Total of two coins collected and two walls hit in the process.

Hints

- Think about Exception Handling

8. Matrix shuffling

Write a program which reads a string matrix from the console and performs certain operations with its elements. User input is provided in a similar way like in the problems above – first you read the **dimensions** and then the **data**.

Your program should then receive commands in format: "**swap row1 col1 row2c col2**" where row1, row2, col1, col2 are **coordinates** in the matrix. In order for a command to be valid, it should start with the "**swap**" keyword along with **four valid coordinates** (no more, no less). You should **swap the values** at the given coordinates (cell [row1, col1] with cell [row2, col2]) **and print the matrix at each step** (thus you'll be able to check if the operation was performed correctly).

If the **command is not valid** (doesn't contain the keyword "swap", has fewer or more coordinates entered or the given coordinates do not exist), print "**Invalid input!**" and move on to the next command. Your program should finish when the string "**END**" is entered.

Examples

Input	Output
2 3 1 2 3 4 5 6 swap 0 0 1 1 swap 10 9 8 7 swap 0 1 1 0 END	5 2 3 4 1 6 Invalid input! 5 4 3 2 1 6
1 2 Hello World 0 0 0 1 swap 0 0 0 1 swap 0 1 0 0 END	Invalid input! World Hello Hello World

Hints

- Think about Exception Handling

9. * Terrorists Win!

On de_dust2 terrorists have planted a bomb (or possibly several of them)! Write a program that sets those bombs off!

A bomb is a string in the format **|...|**. When set off, the bomb destroys all characters inside. The bomb should also destroy **n** characters to the left and right of the bomb. **n** is determined by the **bomb power** (the **last digit of the ASCII sum** of the characters inside the bomb). Destroyed characters should be replaced by '**.**' (dot). For example, we are given the following text:

```
prepare|yo|dong
```

The bomb is **|yo|**. We get the bomb power by calculating the last digit of the sum: **y** (121) + **o** (111) = 23**2**. The bomb explodes and destroys itself and **2** characters to the left and **2** characters to the right. The result is:

```
prepa........ng
```

Input

The input data should be read from the console. On the first and only input line you will receive the text.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The destroyed text should be printed on the console.

Constraints

- The length of the text will be in the range [1…1000].
- The bombs will hold a number of characters in the range [0…100].
- Bombs will not be nested (i.e. bomb inside another bomb).
- Bomb explosions will never overlap with other bombs.
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output
prepare|yo|dong	prepa........ng

Input	Ouput
de_dust2 |A| the best |BB|map!	de_d............bes........p!

10. * Plus-Remove

You are given a sequence of **text lines**, holding symbols, small and capital Latin letters. Your task is to **remove all "plus shapes"** in the text. They may consist of small and capital letters at the same time, or of any symbol. All of the **X shapes** below are **valid**:

a	B	T	p	&	*	etc.
aaa	BBB	TtT	PPp	&&&	***	
a	B	T	P	&	*	

Every **"plus shape"** is 3 by 3 symbols crossing each other on 3 lines. Single **"plus shape"** can be part of **multiple "plus shapes"**. If new **"plus shapes"** are formed after the first removal **you don't have** to remove them.

Input

The input data will be received from the console. It consists of variable number of lines. The input ends with the string "**END**" (it should not be taken into account in the program logic).

Output

Print at the console the input data after removing all **"plus shapes"**.

Constraints

- Each input line will hold between 1 and 100 Latin letters.
- The number of input lines will be in the range [1 ... 100].
- Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

Examples

Input	Output
ab**l5	a**l
bBb*555	*
absh*5	as*
ttHHH	tt
ttth	ttt
END	

Input	Output
888**t*	8***
8888ttt	
888ttt<<	<<
*8*0t>>hi	**0>>hi
END	

Input	Output
@s@a@p@una	@sapuna
p@@@@@@@@dna	p@dna
@6@t@*@*ego	@6tego
vdig*****ne6	vdigne6
li??^*^*	li??^^
END	

11. * String Matrix Rotation

You are given a **sequence of text lines**. Assume these text lines form a **matrix of characters** (pad the missing positions with spaces to build a rectangular matrix). Write a program to **rotate the matrix** by 90, 180, 270, 360, … degrees. Print the result at the console as sequence of strings after receiving the "**END**" command.

Examples

Input	Rotate(90)	Rotate(180)	Rotate(270)
hello softuni exam END h e l l o s o f t u n i e x a m	e s h x o e a f l m t l u o n i	m a x e i n u t f o s o l l e h	i n o u l t m l f a e o x h s e

Input

The input is read from the console:

- The first line holds a command in format "**Rotate(X)**" where **X** are the degrees of the requested rotation.
- The next lines contain the **lines of the matrix** for rotation.
- The input ends with the command "**END**".

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

Print at the console the **rotated matrix** as a sequence of text lines.

Constraints

- The rotation **degrees** is positive integer in the range [0…90000], where **degrees** is **multiple of 90**.
- The number of matrix lines is in the range [1…**1 000**].
- The matrix lines are **strings** of length 1 … 1 000.
- Allowed working time: 0.2 seconds. Allowed memory: 16 MB.

Examples

Input	Output
Rotate(90) hello softuni exam END	esh xoe afl mtl uo n i

Input	Output
Rotate(180) hello softuni exam END	maxe inutfos olleh

Input	Output
Rotate(270) hello softuni exam END	i n ou ltm lfa eox hse

Input	Output
Rotate(720) js exam END	js exam

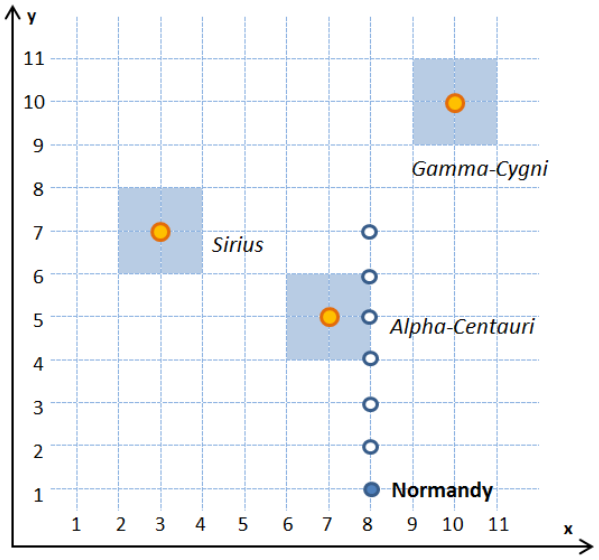
Input	Output
Rotate(810) js exam END	ej xs a m

Input	Output
Rotate(0) js exam END	js exam

12. * To the Stars!

The year is 2185 and the SSR Normandy spaceship explores our galaxy. Unfortunately, the ship suffered severe damage in the last battle with Batarian pirates, and her navigation system is broken. Your task is to write a JavaScript program to help the Normandy safely navigate through the stars back home.

The navigation field is a 2D grid. You are given the names of **3 star systems**, along with **their coordinates(s_x, s_y)** and **the Normandy's initial coordinates(n_x, n_y)**. Assume that a **star's coordinates are in the center of a 2x2 rectangle**. The Normandy **always moves in an upwards direction, 1 unit every turn**. Your task is to inform the Normandy of its current location during its movement.



The Normandy can **only be at one location** at a time. The possible locations are "**<star1 name>**", "**<star2 name>**", "**<star3 name>**" and "**space**". In other words, if the Normandy is in the range of Alpha-Centauri, her location is "alpha-centauri". If she's not in the range of any star system, her location is just "space".

Star systems will **NOT** overlap.

Example: the Normandy's initial location is at (8, 1). There, she in outside of any star system, so she is in "space". She starts moving up and her next two locations at (8, 2) and (8, 3) are again in "space". After that, at (8, 4), (8, 5), (8, 6) she is in the range of Alpha-Centauri – therefore, she is in "alpha-centauri". Her final location (8, 7) is outside any star, and her location is "space".

Input

The input is passed to the first JavaScript function found in your code as **array of several arguments**:

- The first arguments will contain each star system's name and coordinates in the format "<**name**> <**x**> <**y**>", separated by spaces. The **name will be a single word, without spaces**.
- The fourth argument will contain the Normandy's initial coordinates in the format "<**x**> <**y**>", separated by spaces.
- The fifth, last argument, will contain the number **n** – the number of turns the Normandy will be moving.

The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

The output consists of **n + 1** lines – the Normandy's **initial** location, plus the **locations she was in during her movement**, each on a separate line. All locations must be printed **lowercase**.

Constraints

- The grid dimensions will be no larger than 30x30.
- All star systems will be squares with a fixed size: 2x2.
- The turns will be no more than 20.
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output

Input	Output

Sirius 3 7 Alpha-Centauri 7 5 Gamma-Cygni 10 10 8 1 6	space space space alpha-centauri alpha-centauri alpha-centauri space

Terra-Nova 16 2 Perseus 2.6 4.8 Virgo 1.6 7 2 5 4	perseus virgo virgo virgo space